



or truly random, parallel binary machines are an order of magnitude smaller than parallel FSRs generating the same sequence. Therefore, the presented approach can potentially be useful for any application which requires sequences with high spectrum efficiency or high security. Such applications include data transmission, wireless communications, cryptography, and many others [3], [4], [5], [6]. A particularly attractive application is encryption and authentication systems for smartcards and Radio Frequency IDentification (RFID) tags. A low-cost RFID tag can spare only a few hundred gates for security functionality [7]. None of the available cryptographic systems satisfies this requirement at present [8].

The rest of the paper is organised as follows. Section II describes basic notation and definitions used in the sequel. In Section IV, we present an algorithm for constructing an  $m$ -ary machine with the minimum number of stages generating a given  $m$ -ary sequence. In Section V, we show how  $m$ -ary machines can be encoded to generate binary sequences in parallel and demonstrate that such an encoding can be of advantage. Section VI presents the experimental results. Section VII concludes the paper.

## II. PRELIMINARIES

Let  $M = \{0, 1, \dots, m-1\}$ . An  $m$ -ary sequence is vector  $A_m = (a_0, a_1, \dots)$  where  $a_i \in M$  for all  $i \geq 0$ .

If there exist  $k > 0$  and  $k_0 \geq 0$  such that  $a_i = a_{i+k}$  for all  $i \geq k_0$ , then  $A$  is called *eventually* (or *ultimately*) *periodic*. If  $k_0 = 0$ , then  $A$  is called *purely periodic*, or simply *periodic*. The least integers  $k_0$  and  $k$  with this property are called *pre-period* and *period* of the sequence, respectively [9].

For a multiple-valued function  $f : M^n \rightarrow M$ , the  $i$ -set of  $f$  is defined by [10]

$$i\text{-set}(f) = \{x \in M^n : f(x) = i\}.$$

In the binary case, 0-set and 1-set correspond to off-set and on-set of  $f$ , respectively [11].

An  $m$ -ary  $n$ -stage machine consists of  $n$   $m$ -ary storage elements, called *stages*. Each stage  $i \in \{0, 1, \dots, n-1\}$  has an associated *state variable*  $x_i \in M$  which represents the current value of the stage  $i$  and an *updating function*  $f_i : M^n \rightarrow M$  which determines how the value of  $x_i$  is updated.

A *state* of an  $n$ -stage machine is a vector of values of its state variables. At every clock cycle, the next state of a machine is determined from its the current state by updating the values of all stages simultaneously to the values of the corresponding  $f_i$ 's.

The *degree of parallelization* of an  $n$ -stage machine is the number of stages  $p$ ,  $1 < p \leq n$ , which are used to produce its output at each clock cycle.

## III. PREVIOUS WORK

For the case of Linear FSRs (LFSRs), there are two main approaches to constructing an LFSR with the degree of parallelization  $p$ : (1) synthesis of subsequences representing  $p$  decimation of some phase shift of the original LFSR sequence

and (2) computation of the set of states reachable from any state in  $p$  steps.

Let  $S$  be a sequence produced by an LFSR whose characteristic polynomial  $g(x)$  of degree  $n$  is irreducible in  $GF(2)$ . Let  $\alpha$  be a root of  $g(x)$  and let  $T$  be the period of  $S$ . In the method based of synthesis of subsequences [12], the sequence  $S$  is decomposed into  $p$  subsequences  $S_p^j$ , each representing a  $p$  decimation of  $j$ th phase shift of  $S$ . In other words, the  $i$ th element of  $S_p^j$  is equal to  $i \cdot p + j$  element of  $S$ . By Zierler's theorem [13], for  $0 \leq j < p$ , the subsequences  $S_p^j$  can be generated by an LFSR with the following properties:

- The minimum polynomial of  $\alpha^d$  in  $GF(2^n)$  is the characteristic polynomial  $q^*(x)$  of the new LFSR which has:
  - Period  $T^* = T/\gcd(d, T)$ ,
  - Degree  $n^*$ , which is the multiplicative order of 2 in  $Z(T^*)$ .

The Berlekamp-Massey algorithm [14] or its generalizations [15] can be used to find the smallest LFSR for each subsequence  $S_p^j$ . The size of each LFSR is  $n^*$ , which is at most  $n$ , i.e. the overall number of bits in  $p$  LFSRs is at most  $p \times n$ . This method is applicable to any degree of parallelization  $p$  which is not a multiple of the period  $T$ .

The second approach is based on computing the set of states reachable from any state in  $p$  steps. This is usually done by computing  $p$ th power of the connection matrix of the LFSR [16], [17]. Such an approach is applicable to the degrees of parallelization  $1 < p \leq n$ . The size of the register with the degree of parallelization  $p$  in this case is the same as the size of the original LFSR,  $n$ .

For the case of Non-Linear FSRs (NLFSRs), algorithms for finding a shortest NLFSR generating a given binary sequence have been presented in [18], [19], [20], and [9]. An NLFSR with the degree of parallelization  $p$  can be constructed by computing the set of states reachable from any state in  $p$  steps, as in the approach (2) for LFSR. This can be done by computing  $p$ th power of the transition relation of the NLFSR. However, the size of  $p$ th power of the transition relation of an NLFSR usually grows much faster than in the LFSR case. Therefore, in practice, in applications which use NLFSRs with the degree of parallelization  $p$ , NLFSRs are selected so that variables of the  $p$  left-most stages of the NLFSR are not used in the updating functions. In such a case, an NLFSR with the degree of parallelization  $p$  can be constructed by duplicating the updating functions  $p$  times [21], [22], [23].

For binary machines with the degree of parallelization one, an algorithm for constructing a shortest binary machine generating a given binary sequence has been presented in [24].

## IV. SYNTHESIS ALGORITHM

The algorithm presented in this section exploits the property of  $m$ -ary  $n$ -stage machines that *any*  $m$ -ary  $n$ -tuple can be the next state of a given current state. Note that, in the traditional  $n$ -stage NLFSRs in the Fibonacci configuration [1], the next state overlaps with a current state in  $n-1$  positions. NLFSRs in the Galois configuration are more flexible. However,

**Algorithm 1** Construct an  $m$ -ary machine which generates an  $m$ -ary sequence  $A = (a_0, a_1, \dots, a_k)$  with the degree of parallelization one.

---

```

1: for every  $i$  from 0 to  $m - 1$  do
2:    $N_i := 0$ ; /*counts the number of digits with value  $i \in M$ */
3: end for
4: for every  $j$  from 0 to  $k - 1$  do
5:    $N_{a_j} := N_{a_j} + 1$ ;
6: end for
7:  $N_{max} := \max_{i \in M} N_i$ 
8: for every  $i$  from 0 to  $m - 1$  do
9:    $\mathbf{B}_i := \emptyset$ 
10:  for every  $j$  from 0 to  $N_{max} - 1$  do
11:     $\mathbf{B}_i := \mathbf{B}_i \cup \{j * m + i\}$ ;
12:  end for
13: end for
14: for every  $i$  from 0 to  $m - 1$  do
15:    $B_i := [b_{i,0}, b_{i,1}, \dots, b_{i,N_{max}-1}]$  is an arbitrary permutation
      of  $\mathbf{B}_i$ ;
16:    $r_i := 0$ ; /*records how many elements of  $B_i$  were used*/
17: end for
18: for every  $j$  from 0 to  $k - 1$  do
19:    $s_j := b_{a_j, r_{a_j}}$ ; /* $b_{a_j, r_{a_j}}$  is the  $r_{a_j}$ th element of  $B_{a_j}$ */
20:    $r_{a_j} := r_{a_j} + 1$ ;
21: end for
22:  $n = \lceil \log_m N_{max} \rceil + 1$ ;
23: for every  $j$  from 0 to  $k - 1$  do
24:   Expand  $s_j$  as an  $m$ -ary vector  $s_j := (s_{j_{n-1}}, s_{j_{n-2}}, \dots, s_{j_0}) \in M^n$ ;
25: end for
    /*The resulting sequence  $S = (s_0, s_1, \dots, s_{k-1})$  is interpreted
    as a sequence of states of an  $m$ -ary  $n$ -stage machine*/
26: for every  $p$  from 0 to  $n - 1$  do
27:   for every  $i$  from 0 to  $m - 1$  do
28:      $i\text{-set}(f_p) = \emptyset$ ;
29:   end for
30: end for
31: for every  $j$  from 0 to  $k - 1$  do
32:   for every  $p$  from 0 to  $n - 1$  do
33:      $i = s_{(j+1)_p}$ ;
34:      $i\text{-set}(f_p) = i\text{-set}(f_p) \cup \{(s_{j_{n-1}}, s_{j_{n-2}}, \dots, s_{j_0})\}$ ;
35:   end for
36: end for
37: Return  $(f_0, f_1, \dots, f_{n-1})$ ;

```

---

since they do not allow feedforward connections, their set of possible next states is still restricted to a certain subset of all possible states [25].

The input of the algorithm is an  $m$ -ary sequence  $A$  of length  $k$ . First, we show how to construct a sequence of integers  $S = (s_0, s_1, \dots, s_{k-1})$  such that  $s_j \bmod m = a_j$  for all  $j \in \{0, 1, \dots, k-1\}$ . We count the number of occurrences of each of digits with the value  $i \in M$  in  $A$ ,  $N_i$ , and determine

the largest number of occurrences,  $N_{max} = \max_{i \in M} N_i$ .

Let  $\mathbf{B}_i$  be a set consisting of  $N_{max}$  non-negative integers of type  $j \cdot m + i$  for all  $j \in \{0, 1, \dots, N_{max} - 1\}$  and all  $i \in M$ . Let  $B_i = [b_{i,0}, b_{i,1}, \dots, b_{i,N_{max}-1}]$  be an arbitrary permutation of  $\mathbf{B}_i$ .

Initially, for all  $i \in M$ , we set to zero a counter  $r_i$  which counts how many digits of  $B_i$  have been used. Then, for every  $j$  from 0 to  $k - 1$ , we take the  $j$ th element of the sequence  $A$ ,  $a_j$ , and assign  $s_j$  to  $r_{a_j}$ th element of  $B_{a_j}$ . It is easy to see from our construction that  $s_j \bmod m$  is equal to  $a_j$ .

Let  $S = (s_0, s_1, \dots, s_{k-1})$  be a sequence constructed as described above. Each integer  $s_i \in S$  can be represented as an  $m$ -ary expansion  $(s_{i_{n-1}}, s_{i_{n-2}}, \dots, s_{i_0}) \in M^n$  where  $n$  is the number of  $m$ -ary digits needed to represent the largest integer of  $S$  and  $s_{i_0}$  is the least significant digit of the expansion. We interpret each  $n$ -tuple  $(s_{i_{n-1}}, s_{i_{n-2}}, \dots, s_{i_0})$  as a state of an  $m$ -ary  $n$ -stage machine. By construction,  $s_{i_0} = a_i$  for all  $i \in \{0, 1, \dots, k-1\}$ .

Next, we define a mapping  $s_i \mapsto s_{i+1}$ , for all  $i \in \{0, 1, \dots, k-1\}$ , where  $''+''$  is mod  $k$ . This mapping assigns  $s_{i+1}$  to be the next state of a current state  $s_i$  of an  $m$ -ary  $n$ -stage machine. Each of  $m^n - k$  remaining states of the  $m$ -ary  $n$ -stage machine are left unspecified. This gives us a freedom to specify the updating functions in a way which minimizes their circuit complexity.

The  $i$ -sets of the updating functions implementing the resulting mapping are derived as follows. Initially  $i\text{-set}(f_j) = \emptyset$ , for all  $j \in \{0, 1, \dots, n-1\}$  and all  $i \in M$ . For every  $j$  from 0 to  $k-1$ , and every  $p$  from 0 to  $n-1$ , if  $s_{(j+1)_p} \neq 0$ , where  $''+''$  is mod  $k$ , then we add  $(s_{j_{n-1}}, s_{j_{n-2}}, \dots, s_{j_0})$  to the  $i$ -set of  $f_p$  where  $i = s_{(j+1)_p}$ .

The algorithm described above is summarized as Algorithm 1. Its worst-case time complexity is  $O(n \cdot k)$  (assuming  $k > m$  which is normally the case).

*Theorem 1:* The Algorithm 1 constructs an  $m$ -ary  $n$ -stage machine generating an  $m$ -ary sequence  $A$  of length  $k$  with the degree of parallelization one where  $n$  is given by

$$n = \lceil \log_m N_{max} \rceil + 1, \quad (1)$$

where  $N_{max} = \max_{i \in M} N_i$ .

**Proof:** At the step 7 of the Algorithm 1, for each  $i \in M$ ,  $N_i$  equals to the number of digits with the value  $i$  in the sequence  $A$ . From the step 6 of the Algorithm 1 we can conclude that, for each  $i \in M$ , the largest integer  $s_i \in S$  such that  $s_i \bmod m = i$  is equal to  $m(N_i - 1) + i$ . We need  $\lceil \log_m N_i \rceil + 1$   $m$ -ary digits to express this integer for any  $N_i > 0$ . Since  $k > 1$ , the number of stages in the  $m$ -ary  $n$ -stage machine is given by  $\lceil \log_m N_{max} \rceil + 1$  where  $N_{max} = \max_{i \in M} N_i$ . □

The Lemma below shows under which conditions that the bound given by (1) is an exact lower bound.

*Lemma 1:* Given a purely periodic  $m$ -ary sequence  $A_m$  with the period  $k$ , any  $m$ -ary machine which generates  $A_m$  the degree of parallelization one has at least  $n$  stages, where  $n$  is given by (1).

**Proof:** The existence of an  $m$ -ary machine with  $n = \lceil \log_m N_{max} \rceil + 1$  stages which can generate  $A_m$  follows from

the Theorem 1. It remains to prove that no  $m$ -ary  $n'$ -stage machine with  $n' < n$  can generate  $A_m$ .

Assume that such a machine exists. Then, if  $A_m$  is purely periodic and has the period  $k$ , to be able to generate one digit of  $A_m$  per clock cycle with the period  $k$ , the  $m$ -ary  $n'$ -stage machine must have at least  $N_i$  distinct states whose 0th stage has the value  $i$ . We need at least  $\lceil \log_m N_i \rceil + 1$   $m$ -ary stages to implement the largest of these states for any  $N_i > 0$ . So, we can conclude that  $n' \geq \lceil \log_m N_{max} \rceil + 1$  which contradicts the assumption that  $n' < n$ .

□

As an example, consider the 4-ary sequence from the Introduction section:

$$A_4 = (0, 3, 1, 3, 0, 2, 3, 2, 3, 0).$$

We have  $N_{max} = 4$ . So:

$$\begin{aligned} B_0 &= \{0, 4, 8, 12\}, \\ B_1 &= \{1, 5, 9, 13\}, \\ B_2 &= \{2, 6, 10, 14\}, \\ B_3 &= \{3, 7, 11, 15\}. \end{aligned}$$

Suppose we use following permutations of  $B_i$ s:

$$\begin{aligned} B_0 &= [0, 4, 8, 12], \\ B_1 &= [1, 5, 9, 13], \\ B_2 &= [2, 6, 10, 14], \\ B_3 &= [3, 7, 11, 15]. \end{aligned}$$

Then we get:

$$S_4 = (0, 3, 1, 7, 4, 2, 11, 6, 15, 8).$$

Since  $N_{max} = 4$ , from the Theorem 1 we can conclude that the quaternary machine which generates  $A$  has 2 stages. By applying the mapping described in the Algorithm 1 to  $S$ , we get the following  $i$ -sets for the updating functions  $f_0$  and  $f_1$ :

$$\begin{aligned} 0\text{-set}(f_1) &= \{(00), (03), (10), (20)\} \\ 1\text{-set}(f_1) &= \{(01), (13), (23)\} \\ 2\text{-set}(f_1) &= \{(02), (33)\} \\ 3\text{-set}(f_1) &= \{(22)\} \\ 0\text{-set}(f_0) &= \{(13), (20), (33)\} \\ 1\text{-set}(f_0) &= \{(03)\} \\ 2\text{-set}(f_0) &= \{(10), (23)\} \\ 3\text{-set}(f_0) &= \{(00), (01), (02), (12)\}. \end{aligned}$$

The defining tables of these functions are shown in Figure 2. The symbol “-” stands for a don’t care value.

Note that, in Lemma 1, we require that  $A$  is purely periodic with the period  $k$ . The need for the latter condition is obvious: if  $A$  repeats two or more times within the input sequence length  $k$  given to the Algorithm 1, then we need less than eq. (1) stages to generate  $A$ . The former condition is necessary because, in the sequence is eventually periodic, we might be able to generate it with a binary machine with less than eq. (1) stages. As an illustration, consider an eventually periodic

$x_0 \backslash x_1$	0	1	2	3
0	0	0	0	-
1	1	-	-	-
2	2	3	-	-
3	0	1	1	2

Function  $f_1(x_0, x_1)$

$x_0 \backslash x_1$	0	1	2	3
0	3	2	0	-
1	3	-	-	-
2	3	3	-	-
3	1	0	2	0

Function  $f_0(x_0, x_1)$

Fig. 2. Defining table for the updating functions of the 4-ary 2-stage machine in Figure 4(a). The symbol “-” stands for a don’t care (unspecified) value.

$x_{01}x_{00} \backslash x_{11}x_{10}$	00	01	10	11
00	0	0	0	0
01	0	0	0	0
10	1	1	0	0
11	0	0	0	1

Function  $f_{11}(x_{00}, x_{01}, x_{10}, x_{11})$

$x_{01}x_{00} \backslash x_{11}x_{10}$	00	01	10	11
00	1	1	0	0
01	1	0	0	0
10	1	1	0	0
11	0	0	1	0

Function  $f_{01}(x_{00}, x_{01}, x_{10}, x_{11})$

$x_{01}x_{00} \backslash x_{11}x_{10}$	00	01	10	11
00	0	0	0	0
01	1	0	0	0
10	0	1	0	0
11	0	1	1	0

Function  $f_{10}(x_{00}, x_{01}, x_{10}, x_{11})$

$x_{01}x_{00} \backslash x_{11}x_{10}$	00	01	10	11
00	1	0	0	0
01	1	0	0	0
10	1	1	0	0
11	1	0	0	0

Function  $f_{00}(x_{00}, x_{01}, x_{10}, x_{11})$

Fig. 3. Defining tables for the updating functions of the binary 4-stage machine in Figure 4(b) for the case when all don’t cares are specified to 0. The pairs  $(f_{11}, f_{10})$  and  $(f_{01}, f_{00})$  encode the 4-valued functions  $f_1$  and  $f_0$  in Figure 2, respectively.

binary sequence  $(1, 1, 0, 0, 1, 0, 1, 0, 1)$  with pre-period 3 and period 2. By using Algorithm 1, we can construct a binary machine with 4 stages which repeats this sequence with the period 9. However, we can also construct a binary machine with 3 stages whose state transition graph has a cycle of length 2, corresponding to the period  $(0, 1)$  and has a branch implementing  $(1, 1, 0)$  which leads to the cycle. In some cases, the binary machine constructed by the latter approach might be smaller than the one constructed using the Algorithm 1.

## V. GENERATION OF BINARY SEQUENCES

We can use  $m$ -ary  $n$ -stage machines for generating binary sequences by encoding their  $m$ -ary stages and  $m$ -valued functions using at most  $(\lceil \log_2 m \rceil \cdot n)$  binary stages and Boolean functions.

An example, consider the quaternary 2-stage machine from the example in the previous section. Figure 4(a) shows its quaternary implementation. Figure 4(b) shows the same machine in which the updating functions  $f_0$  and  $f_1$  are encoded by a pair of Boolean functions  $(f_{i0}, f_{i1})$ ,  $i \in \{0, 1\}$ , using the encoding  $0 = (00)$ ,  $1 = (01)$ ,  $2 = (10)$ ,  $3 = (11)$ . The defining tables for the Boolean functions are shown in Figure 3. We specified all don’t cares of  $f_0$  and  $f_1$  to 0. The resulting binary 4-stage machine generates the following sequence  $A_2$  two bits per clock cycle:

$$A_2 = (0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0). \quad (2)$$

As we showed in the Introduction, if instead of using quaternary encoding, we use Algorithm 1 to construct a binary machine for  $A_2$  directly, we get  $N_0 = 9$  and  $N_1 = 11$  and thus a machine with  $n = \lceil \log_2 11 \rceil + 1 = 5$  stages.



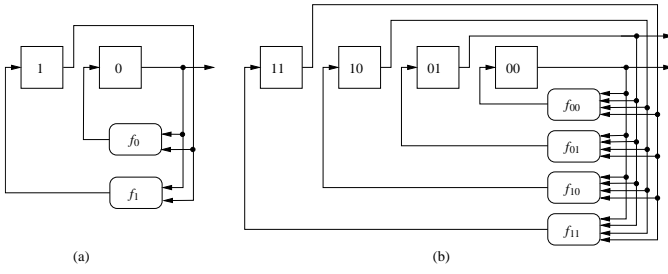


Fig. 4. (a) A quaternary 2-stage machine with the degree of parallelization one. (b) The machine from (a) encoded as a binary 4-stage machine with the degree of parallelization two.

Let us see whether we can reduce the number of stages even more is we use 8-are encoding. We group the bits of  $A_2$  in triples to get the following 8-ary sequence:

$$A_8 = (1, 5, 6, 2, 7, 3, 0).$$

Note that we have added an extra 0 to  $A_2$  to make its length a multiple of 3. Using the Algorithm 1 we can derive the following sequence of integers  $S_8 = (s_0, s_1, \dots, s_7)$  such that  $s_j \bmod 8 = a_j$  for all  $j \in \{0, 1, \dots, 7\}$ :

$$S_8 = (1, 5, 6, 2, 7, 3, 0).$$

As we can see,  $S = A_8$ , because none of the digits of  $A_8$  repeat more than once. By the Theorem 1, we need  $n = \lceil \log_8 1 \rceil + 1 = 1$  stage to implement this sequence by an 8-ary machine. The updating function of this machine is defined is Figure 5. By encoding the 8-ary 1-stage machine in binary, we get a binary 3-stage machine with the updating functions defined in Figure 6 which generates three bits of  $A_2$  per clock cycle. So, we gained one more stage by using the 8-ary encoding.

Before presenting the main result of the paper, let us formally define  $m$ -ary encodings.

**Definition 1:** For  $m = 2^p$ ,  $p > 0$ , an  $m$ -ary encoding of a binary sequence  $A_2$  of length  $k$  is the  $m$ -ary sequence  $A_m$  of length  $\lceil k/p \rceil$  which is obtained from  $A_2$  by replacing the consecutive  $p$ -tuples of bits of  $A_2$ ,  $(a_i, a_{i+1}, \dots, a_{i+p-1})$ ,  $i \in \{0, p, 2p, \dots, \lceil k/p \rceil\}$ , by the value  $a_i \cdot m^{p-1} + a_{i+1} \cdot m^{p-2} + \dots + a_{i+p-1} \cdot m^0$ . If  $k' \bmod p \neq 0$ , then the length of  $A$  is extended to the minimum  $k'$  such that  $k' \bmod p = 0$  and  $k' > k$ . The appended bits are chosen so that the resulting  $N_{max} = \max_{i \in M} N_i$  is minimum.

The following theorems gives the lower bound on the number of stages in binary machine with the degree of parallelization  $p$ .

**Theorem 2:** Let  $A_2$  be a purely periodic binary sequence with the period  $k$ . Any binary machine which generates  $A_2$  with the degree of parallelization  $p \geq 1$  has at least  $n$  stages, where  $n$  is given by:

$$n = \lceil \log_2 N_{max} \rceil + p$$

where  $N_{max} = \max_{i \in M} N_i$  and  $N_i$  is to the number of digits with the value  $i$  in the  $m$ -ary encoding of  $A_2$ ,  $m = 2^p$ .

$x_0$	0	1	2	3	4	5	6	7
	1	5	7	0	-	6	2	3

Fig. 5. Defining table for the updating function  $f_0$  of the 8-ary 1-stage machine from the example.

**Proof:** Let  $m = 2^p$  where  $p$  is the degree of parallelization,  $p > 0$ . From the step 6 of the Algorithm 1 we can conclude that, for each  $i \in M$ , the largest integer  $s_i \in S$  such that  $s_i \bmod m = i$  is equal to  $m(N_i - 1) + i$ . We need  $\lceil \log_2 N_i \rceil + p$  binary digits to express this integer for any  $N_i > 0$ . Therefore, for  $k > 1$ , the number of stages in the binary  $n$ -stage machine is at most  $n \leq \lceil \log_2 N_{max} \rceil + p$  where  $N_{max} = \max_{i \in M} N_i$ .

To be able to generate  $p$  bits of  $A_2$  per clock cycle, the binary  $n$ -stage machine must have at least  $N_i$  distinct states whose  $p$  lest significant bits correspond to the binary encoding of the value  $i$ . If  $A_2$  is purely periodic with the period  $k$ , we need at least  $\lceil \log_2 N_i \rceil + p$  binary stages to implement the largest of these states for any  $N_i > 0$ . Therefore,  $n \geq \lceil \log_2 N_i \rceil + p$ .

So, we can conclude that  $n = \lceil \log_2 N_{max} \rceil + p$ . □

The technique presented above opens a new possibility for increasing the throughput of FSR-based binary sequence generators. As we mentioned in Section III, at present, the generation of  $p$ -bits of a sequence per clock cycle is usually achieved by duplicating the combinatorial logic implementing updating functions of the FSR  $p$  times [21], [22], [23].

As an example, consider the sequence  $A_2$  given by (2). According to the Example V.1 in [9]<sup>1</sup>, the shortest non-linear FSR in the Fibonacci configuration which can generate  $A_2$  has 7 stages and the following updating function of the stage 6:

$$f_6 = \bar{x}_0 \bar{x}_1 \oplus x_0 \bar{x}_1 \oplus \bar{x}_0 x_1 \oplus x_0 x_1 \bar{x}_2 x_3 \oplus \bar{x}_0 x_1 x_2 x_3 \\ \oplus x_0 \bar{x}_1 \bar{x}_2 x_3 \oplus \bar{x}_0 x_1 x_2 x_3 \bar{x}_4 x_5 \bar{x}_6 \oplus x_0 x_1 \bar{x}_2 x_3 x_4 x_5 \bar{x}_6.$$

The updating functions of the remaining stages of the NLFSR are of type  $f_i = x_{i+1}$ , for  $i \in \{0, 1, \dots, 5\}$ . If we use the number of 2-input XORs and ANDs as a measure of cost, then the cost of  $f_6$  is 24 ANDs + 7 XORs.

On the other hand, as shown above, we can generate 3-bits of  $A_2$  per clock cycle using the 3-stage binary machine with the updating functions defined in Figure 6. We can express these functions as follows:

$$f_{02} = x_{00} \bar{x}_{01} \oplus \bar{x}_{00} x_{01} \bar{x}_{02} \\ f_{01} = \bar{x}_{00} x_{01} \oplus x_{00} x_{02} \\ f_{00} = \bar{x}_{02} \oplus x_{00} x_{01}.$$

In total,  $f_{02}$ ,  $f_{01}$  and  $f_{00}$  have 6 AND and 3 XORs. So, the cost of generating 3 bits of  $A_2$  per clock cycle using this binary 3-stage machine is 3 binary stages of a register + 6 ANDs + 3 XORs.

Too make a crude comparison of the two costs, let us assume that the costs of the 2-input AND and the 2-input XOR are 1,

<sup>1</sup>The sequence in the Example V.1 in [9] does not contain the last bit of  $A_2$ , but this does not change the updating functions of the NLFSR.

$x_{02}x_{01}x_{00}$	000	001	010	011	100	101	110	111
	0	1	1	0	0	1	0	0
Function $f_{02}(x_{00}, x_{01}, x_{02})$								
$x_{02}x_{01}x_{00}$	000	001	010	011	100	101	110	111
	0	0	1	0	0	1	1	1
Function $f_{01}(x_{00}, x_{01}, x_{02})$								
$x_{02}x_{01}x_{00}$	000	001	010	011	100	101	110	111
	1	1	1	0	0	0	0	1
Function $f_{00}(x_{00}, x_{01}, x_{02})$								

Fig. 6. Defining tables for the updating functions ( $f_{02}, f_{01}, f_{00}$ ) representing the binary encoding of the 8-valued function in Figure 5 for the case when the don't care is specified to 0.

and the cost of one stage of a register is 2. Then, the cost of the NLFSR is 45, while the cost of the binary machine is 15. So, the binary machine is not only 3 times faster, but also 3 times smaller.

## VI. EXPERIMENTAL RESULTS

To evaluate the presented approach, we compared the areas of binary machines, LFSRs and NLFSRs generating the same sequence for 3 types of sequences: truly random, complementary, and Legendre. All experiments were run on a PC with Intel dual-core 1.8 GHz processor and 2 Gbytes of memory. The area was computed using ABC synthesis tool [26] by first optimizing the circuits with *resyn* script and then by mapping them with *map*. In the results reported below, 1 unit of area is equal to the area of a 2-input NAND gate.

In the first set of experiments, for each  $n$  in the range  $4 \leq n \leq 16$ , we generated 20 truly random sequences of length  $2^n$  using the method [27]. Columns 2-4 of Table I show the areas of the resulting LFSRs, NLFSRs and binary machines (BM) for the degree of parallelization one. Columns 5-7 of Table I shows similar results for the degree of parallelization equal to the number of stages in binary machines (which is always less or equal to the number of stages in LFSRs and NLFSRs). Each entry is an average for 20 sequences.

LFSRs are quite bad for generating truly random sequences.<sup>2</sup> The number of their stages grows roughly as a half of the sequence length. For NLFSRs, the number of stages grows much slower. However, the combinatorial area of parallel NLFSRs grows so fast that they become hard to synthesize for random sequences longer than 256 bits. As we can see from Table I, on average, the area of parallel binary machines is an order of magnitude smaller than the area of parallel LFSRs and NLFSRs.

Table II shows the results for complementary sequences. Complementary sequences are a pair of sequences whose

<sup>2</sup>Note that there is a subset of pseudo-random sequences, called *m*-sequences, for which LFSRs are extremely efficient. An  $n$ -stage LFSR with a primitive polynomial of degree  $n$  generates an *m*-sequence of length  $2^n - 1$ . If the primitive polynomial has  $k$  non-zero terms, then to implement such an LFSR with the degree of parallelization  $p$ , we need  $n$  stages and no more than  $k * p$  XORs. However, due to the linearity of LFSRs *m*-sequences they are easy to reconstruct from a short segment.

aperiodic autocorrelation coefficients sum up to zero [28]. These sequences are known to have a tightly low peak-to-mean envelope power ratio, good error detection capabilities, and high nonlinearity [4]. They are recommended for orthogonal frequency division multiplexing [4] and for multicarrier code division multiple access systems [5]. We can see that, on average, parallel binary machines are an order of magnitude smaller than parallel LFSRs and NLFSRs.

Table III shows the results for extended Legendre sequences. Extended Legendre sequences are known to have the asymptotic merit factor of 6.3421, which is the highest of all known families of sequences of an arbitrary length [6]. The higher the merit factor of a sequence which is used to modulate a signal, the more uniformly the signal energy is distributed over the frequency range. This is important for spread-spectrum communication systems, ranging systems, and radar systems [5], [6]. Again, on average, parallel binary machines are an order of magnitude smaller than parallel LFSRs and NLFSRs.

## VII. CONCLUSION

In this paper, we present a method for constructing binary machines with the minimum number of stages for a given degree of parallelization. Our experimental results show that, for sequences with high linear complexity, such as complementary, Legendre, or truly random sequences, parallel binary machines are an order of magnitude smaller than parallel LFSRs and NLFSRs generating the same sequence.

Our results can be beneficial for any application which requires sequences with high spectrum efficiency or high security, such as data transmission, wireless communications, and cryptography.

## VIII. ACKNOWLEDGMENTS

This work was supported in part by a research grant 621-2010-4388 from the Swedish Research Council.

## REFERENCES

- [1] S. Golomb, *Shift Register Sequences*. Aegean Park Press, 1982.
- [2] G. De Micheli, R. Brayton, and A. Sangiovanni-Vincentelli, "Optimal state assignment for finite state machines," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 4, pp. 269–285, July 1985.
- [3] K. Zeng, C. Yang, D. Wei, and T. R. N. Rao, "Pseudo-random bit generators in stream-cipher cryptography," *Computer*, 1991.
- [4] J. Davis and J. Jedwab, "Peak-to-mean power control in ofdm, golay complementary sequences, and reed-muller codes," *IEEE Transactions on Information Theory*, vol. 45, pp. 2397–2417, Nov. 1999.
- [5] B. Popovic, "Spreading sequences for multicarrier cdma systems-complementary series," *IEEE Transactions on Communications*, vol. 47, pp. 918–926, June 1999.
- [6] R. Kristiansen and M. Parker, "Binary sequences with merit factor  $> 6.3$ ," *IEEE Transactions on Information Theory*, vol. 50, pp. 3385–3389, Dec. 2004.
- [7] A. Juels, "RFID security and privacy: a research survey," *Selected Areas in Communications, IEEE Journal on*, vol. 24, pp. 381–394, Feb. 2006.
- [8] T. Good and M. Benaissa, "ASIC hardware performance," *New Stream Cipher Designs: The eSTREAM Finalists, LNCS 4986*, pp. 267–293, 2008.
- [9] K. Limniotis, N. Kolokotronis, and N. Kalouptsidis, "On the nonlinear complexity and Lempel-Ziv complexity of finite length sequences," *IEEE Transactions on Information Theory*, vol. 53, no. 11, pp. 4293–4302, 2007.

Sequence length	Degree of parallelization = 1			Degree of parallelization = stages in BM			Improvement	
	LFSRs	NLFSRs	BM	LFSRs	NLFSRs	BM	$\frac{a4}{a6}$	$\frac{a5}{a6}$
	$a1$	$a2$	$a3$	$a4$	$a5$	$a6$		
$2^4$	47.35	32.93	72.38	86.3	98.68	20.85	4.73	4.14
$2^5$	104.33	53.4	153	249.48	257.85	41.08	6.28	6.07
$2^6$	218.85	86.15	340	654.65	1007.62	79.03	12.75	8.28
$2^7$	449.03	136.18	724.5	1501.53	4081.98	151.33	26.97	9.92
$2^8$	885.85	236.65	1600.1	3715.9	26638.6	371.43	71.72	10
$2^9$	1910.28	407.3	3258.6	8707.73	-	859.18	-	10.13
$2^{10}$	3889.13	757.95	7306.78	22727.68	-	1759.3	-	12.92
$2^{11}$	8540.27	1399.75	15057.5	-	-	3588.9	-	-
$2^{12}$	15664.25	2567.45	30128.6	-	-	7777.03	-	-
$2^{13}$	30208.38	4765.86	58946.55	-	-	15719.8	-	-
$2^{14}$	-	8817.89	114325.91	-	-	32981.89	-	-
$2^{15}$	-	16084.3	219473.62	-	-	63694.7	-	-
$2^{16}$	-	-	419118.45	-	-	123947.6	-	-

TABLE I

AREA RESULTS FOR RANDOM SEQUENCES (AVERAGE FOR 20 SEQUENCES); '-' STANDS FOR TIME OUT TO COMPUTE THE RESULT (15 MIN).

Sequence length	Degree of parallelization = 1			Degree of parallelization = stages in BM			Improvement	
	LFSRs	NLFSRs	BM	LFSRs	NLFSRs	BM	$\frac{a4}{a6}$	$\frac{a5}{a6}$
	$a1$	$a2$	$a3$	$a4$	$a5$	$a6$		
$2^4$	49	34	81.5	115	155	20.5	5.61	7.56
$2^5$	105	54.5	164	241	411.5	48.5	4.97	8.48
$2^6$	279	92.5	347.5	782	6347.5	91.5	8.55	69.37
$2^7$	493	-	707	1747	-	165.5	10.56	-
$2^8$	1093	-	1486.5	4556	-	470	9.69	-
$2^9$	2161	-	2737	12531	-	909.5	13.78	-
$2^{10}$	4509	-	6348.5	34660	-	1865.5	18.58	-
$2^{11}$	9097	-	11269	82954	-	3874	21.41	-
$2^{12}$	19379	-	23073.5	-	-	8324.5	-	-
$2^{13}$	36951	-	39905	-	-	13888	-	-
$2^{14}$	74089	-	80422.5	-	-	22720.5	-	-
$2^{15}$	-	-	140433	-	-	43094.5	-	-
$2^{16}$	-	-	292710.5	-	-	82670	-	-

TABLE II

AREA RESULTS FOR COMPLEMENTARY SEQUENCES; '-' STANDS FOR TIME OUT TO COMPUTE THE RESULT (15 MIN).

Sequence length	Degree of parallelization = 1			Degree of parallelization = stages in BM			Improvement	
	LFSRs	NLFSRs	BM	LFSRs	NLFSRs	BM	$\frac{a4}{a6}$	$\frac{a5}{a6}$
	$a1$	$a2$	$a3$	$a4$	$a5$	$a6$		
17	42	33	68.5	84	110	19	4.42	5.79
31	97	44.5	146.5	281	192.5	31.5	8.92	6.11
61	231.5	83.5	311	667.5	1248.5	89.5	7.46	13.95
127	482	136.5	640.5	1901	10157.5	180.5	10.53	56.27
257	833	248	1357.5	3115	20787	247	12.61	84.16
557	2144.5	408	2900	9629.5	-	862.5	11.16	-
1021	3796	733	6779	19369	-	1906	10.16	-
2053	8016.5	1356.5	13080	47263.5	-	3652	12.94	-
4099	16358	2596.5	25491.5	-	-	7654	-	-
8233	33930.5	-	50691	-	-	16211	-	-
10223	42422	-	71780	-	-	20160.5	-	-
16127	63012	-	116037	-	-	32423.5	-	-

TABLE III

AREA RESULTS FOR EXTENDED LEGENDRE SEQUENCES; '-' STANDS FOR TIME OUT TO COMPUTE THE RESULT (15 MIN).

- [10] E. Dubrova, "Multiple-valued logic synthesis and optimization," in *Logic Synthesis and Verification*, Eds.: S. Hassoun and T. Sasao, (Kluwer Academic Publishers), pp. 89–114, 2002.
- [11] R. K. Brayton, C. McMullen, G. Hatchel, and A. Sangiovanni-Vincentelli, *Logic Minimization Algorithms For VLSI Synthesis*. Kluwer Academic Publishers, 1984.
- [12] A. Lempel and W. L. Eastman, "High speed generation of maximal length sequences," *IEEE Trans. Comput.*, vol. 20, pp. 227–229, February 1971.
- [13] N. Zierler, "Linear recurring sequences," *Journal of the Society for Industrial and Applied Mathematics*, vol. 2, pp. 31–48, 1959.
- [14] J. L. Massey, "Shift-register synthesis and BCH decoding," *IEEE Transactions on Information Theory*, vol. 15, pp. 122–127, 1969.
- [15] G. L. Feng and K. K. Tzeng, "Algorithm for multisequence shift-register synthesis with applications to decoding cyclic codes," *IEEE Transactions on Information Theory*, vol. 37, no. 5, pp. 1274–1287, 1991.

- [16] I. Goldberg and D. Wagner, "Architectural considerations for cryptanalytic hardware," tech. rep., *Secrets of Encryption Research*, Wiretap Politics and Chip Design, 1998.
- [17] S. Mukhopadhyay and P. Sarkar, "Application of LFSRs for parallel sequence generation in cryptologic algorithms," in *Computational Science and Its Applications - ICCSA 2006*, vol. 3982 of *Lecture Notes in Computer Science*, pp. 436–445, Springer Berlin / Heidelberg, 2006.
- [18] C. J. A. Jansen, "The maximum order complexity of sequence ensembles," in *Proceedings of the 10th International conference on Theory and application of cryptographic techniques*, EUROCRYPT'91, (Berlin, Heidelberg), pp. 153–159, Springer-Verlag, 1991.
- [19] P. Rizomiliotis and N. Kalouptsidis, "Results on the nonlinear span of binary sequences," *IEEE Transactions on Information Theory*, vol. 51, no. 4, pp. 1555–1563, 2005.
- [20] P. Rizomiliotis, N. Kolokotronis, and N. Kalouptsidis, "On the quadratic span of binary sequences," *IEEE Transactions on Information Theory*, vol. 51, no. 5, pp. 1840–1848, 2005.
- [21] M. Hell, T. Johansson, and W. Meier, "Grain - a stream cipher for constrained environments," [citeseer.ist.psu.edu/732342.html](http://citeseer.ist.psu.edu/732342.html).
- [22] C. D. Canniere and B. Preneel, "TRIVIUM specifications," [citeseer.ist.psu.edu/734144.html](http://citeseer.ist.psu.edu/734144.html).
- [23] B. Gittins, H. A. Landman, S. O'Neil, and R. Kelson, "A presentation on VEST hardware performance, chip area measurements, power consumption estimates and benchmarking in relation to the aes, sha-256 and sha-512." Cryptology ePrint Archive, Report 2005/415, 2005. <http://eprint.iacr.org/>.
- [24] E. Dubrova, "Synthesis of binary machines," *IEEE Transactions on Information Theory*, 2011, to appear.
- [25] E. Dubrova, "A transformation from the Fibonacci to the Galois NLF-SRs," *IEEE Transactions on Information Theory*, vol. 55, pp. 5263–5271, November 2009.
- [26] Berkeley Logic Synthesis and Verification Group, "ABC: A system for sequential synthesis and verification, release 70930," 2007.
- [27] D. Rijmenants, "One-time pad," 2011.
- [28] M. J. E. Golay, "Complementary series," *IRE Transactions on Information Theory*, vol. 7, pp. 82–87, Oct. 1961.